NAT: Neural Architecture Transformer for Accurate and Compact Architectures

Yong Guo, Yin Zheng, Mingkui Tan, Qi Chen, Jian Chen, Peilin Zhao, Junzhou Huang

Published in NeurIPS 2019
Contents

1. Background
2. Proposed Method
3. Experimental Results
4. Conclusion
Contents

1. Background

2. Proposed Method

3. Experimental Results

4. Conclusion
Background

Deep neural networks have achieved great success in many computer vision tasks, such as image classification, face recognition, object detection, etc.

Figure: Applications of deep neural networks.
Neural Architecture Design

- Neural architecture design is one of the key factors behind the success of deep neural networks.

- Existing architectures can be divided into two categories:

  1. Hand-crafted architectures
  2. Automatically searched architectures
Hand-crafted Architectures

Several widely used hand-crafted architectures:

- **VGG**
- **ResNet**
- **MobileNetV2**

**Figure:** Examples of hand-crafted architectures.

**Limitations of hand-crafted architecture design process**

- Hand-crafted methods rely on **substantial human expertise**.
- Hand-crafted methods cannot fully **explore the whole architecture space**.
Automatically Searched Architectures

- There is a growing interest to automate the manual process of architecture design by Neural Architecture Search (NAS).

Figure: Examples of NAS based architectures.

Limitations of NAS methods

- Search space is extremely large, e.g., billions of candidate architectures.
- NAS methods may find suboptimal architectures with limited performance.
Architecture Optimization

Since both the hand-crafted and NAS based architectures are not optimal, can we optimize architectures to obtain the better ones?

- One can design architecture optimization methods to optimize existing architectures for better performance.

Figure: Architecture optimization scheme.
Existing Architecture Optimization Methods

- Neural Architecture Optimization (NAO)

**Limitations of NAO**

- NAO often produces a **totally different architecture** from the input one.
- NAO may introduce extra parameters or additional computational cost.

*Figure: Framework of Neural Architecture Optimization (NAO).*
Contents

1. Background

2. Proposed Method

3. Experimental Results

4. Conclusion
Motivation

- Both hand-crafted architectures and NAS based architectures may contain non-significant or redundant operations.
- Existing architecture optimization methods may introduce extra parameters or additional computational cost into the architectures.

How to transform the redundant operations in any arbitrary architecture to improve the performance without introducing extra computational cost?
Problem Definition

Our goal: Transforming any arbitrary architecture for better performance and less computational cost.

One solution: Replacing the redundant operations with the more efficient ones.

We divide the operations into three categories \( \{S, N, O\} \). \( S \) denotes skip connection, \( N \) denotes null connection, \( O \) denotes the other operations.

- We have \( c(O) > c(S) > c(N) \), where \( c(\cdot) \) is a function to evaluate the computational cost.
- To reduce the computational cost, we allow the transitions: \( O \rightarrow S, O \rightarrow N, S \rightarrow N \).
- Since skip connection has negligible cost but often can significantly improve the performance, we also allow \( N \rightarrow S \).
Optimization for Single Architecture

Given a specific architecture $\hat{\beta}$, we seek to find the optimal architecture $\alpha$. Then, the optimization problem can be formulated as

$$\max_{\alpha} R(\alpha | \hat{\beta}), \text{ s.t. } c(\alpha) \leq \kappa$$

- $R(\alpha | \hat{\beta}) = R(\alpha, w_\alpha) - R(\hat{\beta}, w_{\hat{\beta}})$ denotes the performance difference between the optimized architectures $\alpha$ and the given architectures $\hat{\beta}$. $w_\alpha$ and $w_{\hat{\beta}}$ are the parameters of $\alpha$ and $\hat{\beta}$.
- $c(\cdot)$ is a function to measure the computation cost of architectures.
- $\kappa$ is an upper bound of the computational cost.
Optimization for Arbitrary Architecture

- The previous optimization problem is only concentrated on one specific $\hat{\beta}$.
- We hope to find a transformer to optimize any arbitrary architecture $\beta \sim p(\cdot)$.

To this end, we seek to train the transformer parameterized $\theta$ by optimizing the objective

$$\max_{\theta} \mathbb{E}_{\beta \sim p(\cdot)} \left[ \mathbb{E}_{\alpha \sim \pi(\cdot | \beta; \theta)} R(\alpha | \beta) \right], \text{ s.t. } c(\alpha) \leq \kappa, \alpha \sim \pi(\cdot | \beta; \theta)$$

where $\mathbb{E}_{\beta \sim p(\cdot)} \left[ \mathbb{E}_{\alpha \sim \pi(\cdot | \beta; \theta)} R(\alpha | \beta) \right]$ denotes the expectation of $R(\alpha | \beta)$ over the distribution of $\beta \sim p(\cdot)$ and the distribution of $\alpha \sim \pi(\cdot | \beta; \theta)$. 
Optimization for Arbitrary Architecture

\[
\max_{\theta} \mathbb{E}_{\beta \sim p(\cdot)} \left[ \mathbb{E}_{\alpha \sim \pi(\cdot | \beta; \theta)} R(\alpha | \beta) \right], \text{ s.t. } c(\alpha) \leq \kappa, \alpha \sim \pi(\cdot | \beta; \theta)
\]

Several challenges regarding the optimization problem

- It is hard to find a comprehensive measure to accurately evaluate the cost.
- The upper bound of computational cost \(\kappa\) is hard to determine.
- How to compute \(\mathbb{E}_{\beta \sim p(\cdot)} \left[ \mathbb{E}_{\alpha \sim \pi(\cdot | \beta; \theta)} R(\alpha | \beta) \right]\) still remains a question.
Our solution

- We cast the optimization problem into a Markov decision process (MDP).
- We seek to make a series of decisions to replace redundant operations with the more computationally efficient operations.

Benefits: We do not have to evaluate the cost $c(\alpha)$ or determine the upper bound $\kappa$ to obtain an architecture with less computational cost.
Details of MDP

- An **architecture** is defined as a **state**.
- A **transformation mapping** $\beta \rightarrow \alpha$ is defined as an **action**.
- The **accuracy improvement** on validation set is regraded as **reward**.
- The policy $\pi(\cdot | \beta; \theta)$ parameterized by $\theta$ is the **probability distribution of the action**.

Based on MDP, how to build a model to learn the **optimal policy** $\pi$?
Policy Learning by Graph Convolution Networks

Graph Representation of Architectures: We use the directed acyclic graph (DAG) to represent both hand-crafted architectures and NAS based architectures.

- Node: feature maps of a specific layer in deep networks
- Edge: a computational module or operation, e.g., convolution or max pooling

![Network View](a) network view

![Graph View](b) graph view

**Figure:** Graph representation examples of hand-crafted architectures.

**Figure:** Graph representation examples of NAS based architectures.
To better exploit the adjacency information of the operations in an architecture, we use a two-layer graph convolutional network (GCN) to build the controller.

\[ Z = f(X, A) = \text{Softmax} \left( \mathbf{A} \sigma \left( \mathbf{A} \mathbf{X} \mathbf{W}^{(0)} \right) \mathbf{W}^{(1)} \mathbf{W}^{\text{FC}} \right) \]

**Notations**
- **\( A \)**: the adjacency matrix of the architecture graph.
- **\( X \)**: the attributes of the nodes together with their two input edges in the graph.
- **\( \mathbf{W}^{(0)} \) and \( \mathbf{W}^{(1)} \)**: the weights of two graph convolution layers.
- **\( \mathbf{W}^{\text{FC}} \)**: the weight of the fully-connected layer, \( \sigma \) is a non-linear activation function.
- **\( Z \)**: the probability distribution of different candidate operations, i.e., the learned policy.
Training Method

We train the transformer parameters $\theta$ and the model parameter $w$ in an alternative way.

- Training the model parameters $w$:
  
  $$w \leftarrow w - \eta \frac{1}{m} \sum_{i=1}^{m} \nabla_w \mathcal{L}(\beta_i, w)$$

  where $\mathcal{L}(\cdot)$ is the cross-entropy loss, $\eta$ is the learning rate.

- Training the transformer parameters $\theta$:
  
  To encourage exploration, we introduce an entropy regularization term:
  
  $$J(\theta) = \mathbb{E}_{\beta \sim p(\cdot)} \left[ \mathbb{E}_{\alpha \sim \pi(\cdot|\beta;\theta)} \left[ R(\alpha, w) - R(\beta, w) \right] + \lambda H(\pi(\cdot|\beta;\theta)) \right]$$

  $$= \sum_{\beta} p(\beta) \left[ \sum_{\alpha} \pi(\alpha|\beta;\theta) (R(\alpha, w) - R(\beta, w)) + \lambda H(\pi(\cdot|\beta;\theta)) \right]$$

  where $H(\cdot)$ evaluates the entropy of the policy, and $\lambda$ controls the strength of the entropy regularization term.
Algorithm 1 Training method for Neural Architecture Transformer (NAT).

Require: The number of sampled input architectures in an iteration $m$, the number of sampled optimized architectures for each input architecture $n$, learning rate $\eta$, regularizer parameter $\lambda$, input architecture distribution $p(\cdot)$, shared model parameters $w$, transformer parameters $\theta$.

1: Initiate $w$ and $\theta$.
2: while not convergent do
3:   for each iteration on training data do
4:     // Fix $\theta$ and update $w$.
5:     Sample $\beta_i \sim p(\cdot)$ to construct a batch $\{\beta_i\}_{i=1}^m$.
6:     Update the model parameters $w$ by descending the gradient:
7:     \[ w \leftarrow w - \eta \frac{1}{m} \sum_{i=1}^m \nabla_w \mathcal{L}(\beta_i, w). \]
8:   end for
9:   for each iteration on validation data do
10:      // Fix $w$ and update $\theta$.
11:     Sample $\beta_i \sim p(\cdot)$ to construct a batch $\{\beta_i\}_{i=1}^m$.
12:     Obtain $\{\alpha_j\}_{j=1}^n$ according to the policy learned by GCN.
13:     Update the parameters $\theta$ by descending the gradient:
14:     \[ \theta \leftarrow \theta - \eta \frac{1}{mn} \sum_{i=1}^m \sum_{j=1}^n \left[ \nabla_{\theta} \log \pi(\alpha_j | \beta_i; \theta) \left( R(\alpha_j, w) - R(\beta_i, w) \right) + \lambda \nabla_{\theta} \mathcal{H}(\pi(\cdot | \beta_i; \theta)) \right]. \]
15:   end for
16: end while
Contents

1. Background

2. Proposed Method

3. Experimental Results

4. Conclusion
Visual Results of Hand-crafted Architectures

- Results on several hand-crafted architectures, including VGG, ResNet, and MobileNet.
**Visual Results of Hand-crafted Architectures**

- Results on several hand-crafted architectures, including VGG, ResNet, and MobileNet.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>View of Graph</th>
<th>View of Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNetV2</td>
<td><img src="image1" alt="MobileNetV2 Graph" /></td>
<td><img src="image2" alt="MobileNetV2 Network" /></td>
</tr>
<tr>
<td>NAT-MobileNetV2</td>
<td><img src="image3" alt="NAT-MobileNetV2 Graph" /></td>
<td><img src="image4" alt="NAT-MobileNetV2 Network" /></td>
</tr>
</tbody>
</table>

- NAT introduces additional skip connections to improve the performance.
Comparison on Hand-crafted Architectures

- Results on several hand-crafted architectures, including VGG, ResNet, and MobileNet.

<table>
<thead>
<tr>
<th>Model</th>
<th>Method</th>
<th>#Params (M)</th>
<th>#MAdds (M)</th>
<th>Acc. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR-10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VGG16</td>
<td>NAO [31]</td>
<td>19.5</td>
<td>548</td>
<td>95.72</td>
</tr>
<tr>
<td></td>
<td>NAT</td>
<td>15.2</td>
<td>313</td>
<td>96.04</td>
</tr>
<tr>
<td>ResNet20</td>
<td>/</td>
<td>0.3</td>
<td>41</td>
<td>91.37</td>
</tr>
<tr>
<td></td>
<td>NAO [31]</td>
<td>0.4</td>
<td>61</td>
<td>92.44</td>
</tr>
<tr>
<td></td>
<td>NAT</td>
<td>0.3</td>
<td>41</td>
<td>92.95</td>
</tr>
<tr>
<td>ResNet56</td>
<td>/</td>
<td>0.9</td>
<td>127</td>
<td>93.21</td>
</tr>
<tr>
<td></td>
<td>NAO [31]</td>
<td>1.3</td>
<td>199</td>
<td>95.27</td>
</tr>
<tr>
<td></td>
<td>NAT</td>
<td>0.9</td>
<td>127</td>
<td>95.40</td>
</tr>
<tr>
<td>MobileNetV2</td>
<td>/</td>
<td>2.3</td>
<td>91</td>
<td>94.47</td>
</tr>
<tr>
<td></td>
<td>NAO [31]</td>
<td>2.9</td>
<td>131</td>
<td>94.75</td>
</tr>
<tr>
<td></td>
<td>NAT</td>
<td>2.3</td>
<td>91</td>
<td>95.17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>Method</th>
<th>#Params (M)</th>
<th>#MAdds (M)</th>
<th>Acc. (%)</th>
<th>Top-1</th>
<th>Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImageNet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VGG16</td>
<td>NAO [31]</td>
<td>147.7</td>
<td>18896</td>
<td>72.9</td>
<td>91.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NAT</td>
<td>138.4</td>
<td>15620</td>
<td>71.6</td>
<td>92.0</td>
<td></td>
</tr>
<tr>
<td>ResNet18</td>
<td>/</td>
<td>11.7</td>
<td>1580</td>
<td>69.8</td>
<td>89.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NAO [31]</td>
<td>17.9</td>
<td>2246</td>
<td>70.8</td>
<td>89.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NAT</td>
<td>11.7</td>
<td>1580</td>
<td>71.1</td>
<td>90.0</td>
<td></td>
</tr>
<tr>
<td>ResNet50</td>
<td>/</td>
<td>25.6</td>
<td>3530</td>
<td>76.2</td>
<td>92.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NAO [31]</td>
<td>34.8</td>
<td>4505</td>
<td>77.4</td>
<td>93.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NAT</td>
<td>25.6</td>
<td>3530</td>
<td>77.7</td>
<td>93.5</td>
<td></td>
</tr>
<tr>
<td>MobileNetV2</td>
<td>/</td>
<td>3.4</td>
<td>300</td>
<td>72.0</td>
<td>90.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NAO [31]</td>
<td>4.5</td>
<td>513</td>
<td>72.2</td>
<td>90.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NAT</td>
<td>3.4</td>
<td>300</td>
<td>72.5</td>
<td>91.0</td>
<td></td>
</tr>
</tbody>
</table>

- NAT based models yield significantly better performance with approximately the same computational cost as the baseline models.
Visual Results on NAS based Architectures

- Results on several NAS based architectures, including ENAS, DARTS, and NAONet.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Normal cell</th>
<th>Reduction cell</th>
<th>Architecture</th>
<th>Normal cell</th>
<th>Reduction cell</th>
</tr>
</thead>
<tbody>
<tr>
<td>DARTS</td>
<td><img src="image" alt="DARTS Normal Cell" /></td>
<td><img src="image" alt="DARTS Reduction Cell" /></td>
<td>ENAS</td>
<td><img src="image" alt="ENAS Normal Cell" /></td>
<td><img src="image" alt="ENAS Reduction Cell" /></td>
</tr>
<tr>
<td>NAT-DARTS</td>
<td><img src="image" alt="NAT-DARTS Normal Cell" /></td>
<td><img src="image" alt="NAT-DARTS Reduction Cell" /></td>
<td>NAT-ENAS</td>
<td><img src="image" alt="NAT-ENAS Normal Cell" /></td>
<td><img src="image" alt="NAT-ENAS Reduction Cell" /></td>
</tr>
</tbody>
</table>
Visual Results on NAS based Architectures

- Results on several NAS based architectures, including ENAS, DARTS, and NAONet.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Normal cell</th>
<th>Reduction cell</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAONet</td>
<td><img src="image1" alt="Diagram" /></td>
<td><img src="image2" alt="Diagram" /></td>
</tr>
<tr>
<td>NAT-NAONet</td>
<td><img src="image3" alt="Diagram" /></td>
<td><img src="image4" alt="Diagram" /></td>
</tr>
</tbody>
</table>
Comparison on NAS based Architectures

- Results on several NAS based architectures, including ENAS, DARTS, and NAONet.

<table>
<thead>
<tr>
<th>Model</th>
<th>Method</th>
<th>#Params (M)</th>
<th>#MAdds (M)</th>
<th>Acc. (%)</th>
<th>Model</th>
<th>Method</th>
<th>#Params (M)</th>
<th>#MAdds (M)</th>
<th>Acc. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AmoebaNet† [34]</td>
<td>3.2</td>
<td>-</td>
<td>96.73</td>
<td></td>
<td>AmoebaNet [34]</td>
<td>5.1</td>
<td>555</td>
<td>74.5</td>
<td>92.0</td>
</tr>
<tr>
<td>PNAS† [28]</td>
<td>3.2</td>
<td>-</td>
<td>96.67</td>
<td></td>
<td>PNAS [28]</td>
<td>5.1</td>
<td>588</td>
<td>74.2</td>
<td>91.9</td>
</tr>
<tr>
<td>SNAS† [48]</td>
<td>2.9</td>
<td>-</td>
<td>97.08</td>
<td></td>
<td>SNAS [48]</td>
<td>4.3</td>
<td>522</td>
<td>72.7</td>
<td>90.8</td>
</tr>
<tr>
<td>GHN† [52]</td>
<td>5.7</td>
<td>-</td>
<td>97.22</td>
<td></td>
<td>GHN [52]</td>
<td>6.1</td>
<td>569</td>
<td>73.0</td>
<td>91.3</td>
</tr>
<tr>
<td>ENAS† [33]</td>
<td>NAT</td>
<td>4.6</td>
<td>804</td>
<td>97.11</td>
<td>ENAS [33]</td>
<td>NAT</td>
<td>5.6</td>
<td>679</td>
<td>73.8</td>
</tr>
<tr>
<td></td>
<td>NAO [31]</td>
<td>4.5</td>
<td>763</td>
<td>97.05</td>
<td></td>
<td>NAO [31]</td>
<td>5.5</td>
<td>656</td>
<td>73.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.6</td>
<td>804</td>
<td>97.24</td>
<td></td>
<td></td>
<td>5.6</td>
<td>679</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>73.9</td>
</tr>
<tr>
<td>DARTS† [29]</td>
<td>NAT</td>
<td>3.3</td>
<td>533</td>
<td>97.06</td>
<td>DARTS [29]</td>
<td>NAT</td>
<td>5.9</td>
<td>595</td>
<td>73.1</td>
</tr>
<tr>
<td></td>
<td>NAO [31]</td>
<td>3.5</td>
<td>577</td>
<td>97.09</td>
<td></td>
<td>NAO [31]</td>
<td>6.1</td>
<td>627</td>
<td>73.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.0</td>
<td>483</td>
<td>97.28</td>
<td></td>
<td></td>
<td>3.9</td>
<td>515</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NAONet† [31]</td>
<td>NAO [31]</td>
<td>128</td>
<td>66016</td>
<td>97.89</td>
<td>NAONet [31]</td>
<td>NAO [31]</td>
<td>11.35</td>
<td>1360</td>
<td>74.3</td>
</tr>
<tr>
<td></td>
<td>NAT</td>
<td>143</td>
<td>73705</td>
<td>97.91</td>
<td></td>
<td></td>
<td>11.83</td>
<td>1417</td>
<td>74.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>113</td>
<td>58326</td>
<td>98.01</td>
<td></td>
<td></td>
<td>8.36</td>
<td>1025</td>
<td>74.8</td>
</tr>
</tbody>
</table>

- NAT based models yield significantly better performance with less or comparable computational cost as the baseline models.
Comparison of Different Policy Learners

- We compare several policy learners, including Random Search, LSTM, and two GCN based methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>VGG16</th>
<th>ResNet20</th>
<th>MobileNetV2</th>
<th>ENAS$^\dagger$</th>
<th>DARTS$^\dagger$</th>
<th>NAONet$^\dagger$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Search</td>
<td>93.17</td>
<td>91.56</td>
<td>94.38</td>
<td>96.58</td>
<td>95.17</td>
<td>96.31</td>
</tr>
<tr>
<td>LSTM</td>
<td>94.45</td>
<td>92.19</td>
<td>95.01</td>
<td>97.05</td>
<td>97.05</td>
<td>97.93</td>
</tr>
<tr>
<td>Maximum-GCN</td>
<td>94.37</td>
<td>92.57</td>
<td>94.87</td>
<td>96.92</td>
<td>97.00</td>
<td>97.90</td>
</tr>
<tr>
<td>Sampling-GCN (Ours)</td>
<td><strong>95.93</strong></td>
<td><strong>92.97</strong></td>
<td><strong>95.13</strong></td>
<td><strong>97.21</strong></td>
<td><strong>97.26</strong></td>
<td><strong>97.99</strong></td>
</tr>
</tbody>
</table>

- Our Sampling-GCN method significantly outperforms the other methods.
Contents

1. Background

2. Proposed Method

3. Experimental Results

4. Conclusion
We propose a novel Neural Architecture Transformers (NAT) to optimize any arbitrary architectures for better performance without extra computational cost.

We cast the problem into a Markov decision process (MDP) and employ graph convolutional network (GCN) to learn the optimal policy.

Extensive experiments show the effectiveness of NAT on both hand-crafted and NAS based architectures.
Thanks!
Q & A